Active Request Monitor

Local Application

Tue, Mar 7, 2006

The local application REQM is used to monitor active data requests, including those of both Classic and Acnet protocols. The objective is to determine whether replies to a requesting node are falling on deaf ears, say, because the node "goes away." If REQM reaches the conclusion that the reply data is no longer needed, it cancels the request. This note describes the functionality of REQM and how it works.

Active request lists

Local application REQM monitors active data requests by examining the linked list of active request blocks maintained by the underlying system code. At 1 Hz, it builds two lists of such requests within its allocated static memory block. One that is located at offset 0x0080 has records (maximum 32) of the following format:

Field	*Size*	*Meaning*
tNode	2	requesting node#
nReq	2	#active requests
tryCnt	1	retry count
ageCnt	1	time-out counter
mBType	2	request block type#

As indicated, this list keeps track of all active requesting nodes and how many active requests there are from each. It includes the fields for timing out queries (described later) that are designed to determine whether the requester is still active. The request block type# allows distinguishing Classic and Acnet protocols.

The above reference to "requesting nodes" is slightly incomplete. A node# is nearly always a 16-bit pseudo node number that carries a reference to an IPARP table entry that includes both the IP address of the requesting node and the UDP source port# that is used to target replies. Such a source port can have many active requests, each distinguished by a request id that was included in the request message header and is returned in the corresponding reply message headers. (The request id for Classic protocol is often called a "list number.")

The other list of requests maintained by REQM is at offset 0x0800 in its static memory block. Its records (maximum 128) are also 8 bytes in size, as follows:

Field	*Size*	*Meaning*
fNode	2	requesting node# (friendly format)
nBytes	2	#bytes returned in each reply message
nDev	2	#devices requested
ftd	2	frequency-time-descriptor for Acnet RETDAT requests

There is one record for each active request encountered in the linked list of active requests. This list is maintained as a diagnostic that is not strictly needed by REQM to perform its monitoring function. It can be most easily listed by using the "Print Memory" page application PMEM, specifying a target "address" of "REQM+800". The meaning of this format for PMEM is the contents of the REQM static memory block at offset 0x800. (It uses listype #96 to acquire this data. For more details on this, see the note, *LA Static Memory Access*.)

The meaning of a "friendly" node# that is shown in place of the usual pseudo node# is that the native node# is used when possible, along with an indication (in the high nibble) of

whether the protocol is Acnet or Classic. Examples for `node0509` are `0xC509` for the Classic protocol and `0xA509`, or `0xA9B6`, for the Acnet protocol. (Its native node# is `0x0509`, and its Acnet node# is `0x09B6`, for trunk 9, node 182.) An example of an Acnet console client `cns07` might be `0xA907`. The logic affording this translation of pseudo node# to friendly node# is included in the system code, accessible via listype #95. It relies on the UDP source port matching the standard 6800 for Classic and 6801 for Acnet. (For other ports, a pseudo node# is used; there is no friendly node equivalent.) The `REQM` code builds a local (Classic) request using this listype when it needs the translation and calls for immediate results, which works because it is a local request. For more details on this, see the note, *Pseudo Node Translation*.

The `nBytes`, `nDev`, and `ftd` fields are taken from the request message header in the case of Acnet `RETDAT` requests. For other Acnet requests, the `nBytes` field is set to the value `0xB000 + blk`, where `blk` is the request block type#, and the other two fields are set to zero. (The main "other" Acnet protocol is the `FTPMAN` protocol, which will be shown as `0xB011`.)

For Classic protocol requests, the `nBytes` field represents the total number of data bytes in the replies; the `nDev` field is the number of longwords in the internal pointers array, which is normally the number of listypes times the number of idents; and the `ftp` field is the period in 15 Hz cycles, for periodic requests, or (`0x8000` + `evtNum`) for clock event-based requests.

Monitoring logic

In order to monitor whether a requesting node is alive, the local `NETFRAME` data stream records are monitored. This data stream is used to record each datagram received by, or transmitted from, the local node. Specifically, this allows `REQM` to monitor whether any messages are received by the requesting node that would confirm its continued existence. Considering each requesting node separately, if a period of time, normally set for 28 seconds, passes without hearing anything from that node, `REQM` generates a query designed to elicit a response from that node. If a second period of time passes, and there still has not been any datagram seen from that node, another attempt is made. Only after 3 attempts, and 4 periods of time normally amounting to nearly 2 minutes, will `REQM` determine that the requester is no longer alive. And it will therefore cancel all active requests for which replies target that node. All this monitoring logic is desirable since both Classic and Acnet are based on the UDP (connectionless) transport protocols. A single request in either protocol can prompt multiple replies without end until the request is canceled.

Querying a requesting node

Consider the Acnet protocols first. To prompt a response from an Acnet client, an Acnet request message is sent to the client node specifying a null destination task name (`0x00000000`). This should elicit an Acnet error status message that means "no such task." All that matters is that `REQM` will soon notice that the requesting node talked to the local node. Note that the query only checks for the Acnet task being alive, not actually for the client task. But it confirms that there still is a valid network connection to Acnet at that node.

As for Classic protocols, one could imagine sending a canned request message to elicit a reply from a requesting node. But Classic clients are not required to support interpretation of Classic requests. They often are only designed to send a Classic request and interpret Classic replies. So a different method is used, that of sending a reply message that carries an invalid message id, specifically `0x7FF`. (Classic message ids are 11 bits, so this is the maximum value, one that, in order for `REQM` to work properly, should not be used by a Classic client.) The response by a Classic client to any reply for which the message id is unknown is to send a cancel message to announce that no more such replies are desired. This scheme allows `REQM` to verify the continued health of a Classic client requester.

Query details

Querying a node via an Acnet request is easy. The Network Layer routines, including `NetQueue` and `NetSend`, provide that support.

Classic protocol support does not easily lend itself to emitting a reply message, so `REQM` makes up a one-shot Classic request message on behalf of the requesting node, using list number `0x7FF`, passing it to the message queue on which the `Classic` task awaits. In the normal course of events, then, the `Classic` task discovers this message as one received from the requesting node. It interprets the request and quickly emits the reply to that node. This reply should then provoke that requesting node to return a cancel message.

When query fails

If a Classic or Acnet client fails to return a reply, the typical symptom observed is that an active request remains active for less than 2 minutes—again assuming the usual 28 second period parameter is being used for `REQM` in the target node. If a user reports this characteristic, one should look at the first list maintained by `REQM` at offset `0x0080`. (An easy way to do this is via the Memory Dump page application `MDMP`, after obtaining the base address of the `REQM` static memory block from page application `LAPP`, usually attached to Page `E` on a "little console.") A requesting node record will be seen to advance the `ageCnt` field by 1 each second, until 28 is reached, then `tryCnt` will advance from 0 to 1, and the `ageCnt` is reset to commence counting up again. Under normal conditions, the `tryCnt` is promptly reset again. But under under failure conditions, the `tryCnt` is not reset, and after another 28 seconds, the `tryCnt` will advance to 2, and so on. Eventually, after the `tryCnt` reaches 3 and remains there for 28 seconds, the entry will disappear, as the request has been internally canceled.

When `REQM` decides to cancel all requests from a requesting node, it calls the system routine `CanChain` to do it. That routine cancels all Acnet and Classic requests from a given node. Again, note that "node" here may merely be one UDP source port of a (real) node.

Miscellany

At this writing, the IRM version of `REQM` has been in use for about 10 years. It could be simplified somewhat, as we no longer require support for token ring nor for raw ethernet. All data request networking is now IP-based on ethernet.

Because this request monitor functionality is provided by a local application, it is possible to turn it off by disabling `REQM`. But it is normally installed and enabled in every front end. As a diagnostic, the list of active requests maintained at offset `0x0800` in the `REQM` static memory block can often be quite useful.

The monitoring is done by `REQM` in a front end node to assure that a requesting node is still alive. From the opposite point-of-view, one can ask about the client side monitoring the continued health of the replying node. The logic that does this is part of the system code, in which a "server node," acting on behalf of a requesting node client, determines that a contributing node for a given request is not returning any data replies. After the server node's patience is exhausted, it resends the request to the contributing node, hoping to remind it of the request in which its participation is expected. (Perhaps it is rebooting.) It will continue to do this every 2 seconds as long as the request has not been canceled by the requesting node. Details of this logic are covered in a number of other notes.